Package ‘EasyABC’
February 19, 2015

Type     Package
Title    Efficient Approximate Bayesian Computation Sampling Schemes
Version  1.4
Date     2014-12-16
Author   Franck Jabot, Thierry Faure, Nicolas Dumoullin
Maintainer Nicolas Dumoulin <nicolas.dumoulin@irstea.fr>
URL      http://easyabc.r-forge.r-project.org/
Depends  R (>= 2.14.0), abc
Imports  pls, mnormt, MASS, parallel, lhs
Description Enables launching a series of simulations of a computer code from the R session, and to retrieve the simulation outputs in an appropriate format for post-processing treatments. Five sequential sampling schemes and three coupled-to-MCMC schemes are implemented.
License  GPL-3
LazyLoad yes
Repository CRAN
Repository/R-Forge/Project easyabc
Repository/R-Forge/Revision 409
Repository/R-Forge/DateTimeStamp 2014-12-17 08:01:53
Date/Publication 2014-12-17 10:45:18
NeedsCompilation yes

R topics documented:

- EasyABC-package .................................................. 2
- ABC_emulation .................................................... 2
- ABC_mcmc .......................................................... 5
- ABC_rejection .................................................... 10
- ABC_sequential ................................................... 14
- binary_model ...................................................... 21
- binary_model_cluster ............................................ 23
- trait_model ....................................................... 24
Description

The package EasyABC enables to launch a series of simulations of a computer code from the R platform, and to retrieve the simulation outputs in an appropriate format for post-processing treatments. Four sequential sampling schemes and three coupled-to-MCMC schemes are implemented. EasyABC further enables to launch the simulations in parallel on multiple cores of a computer.

Details

Package: EasyABC Type: Package Version: 1.4 Date: 2014-12-16 License: GPL-3 LazyLoad: yes

Author(s)

Franck Jabot, Thierry Faure, Nicolas Dumoulin

See Also

ABC_rejection, ABC_sequential, ABC_mcmc, binary_model, binary_model_cluster

Description

This function launches a series of nb_design_pts model simulations with model parameters drawn in the prior distribution specified in prior_matrix, build an emulator with these computed design points and then launches a series of nb_simul emulator simulations.

Usage

```
ABC_emulation(model, prior, nb_design_pts, nb_simul, prior_test=NULL, summary_stat_target=NULL, emulator_span = 50, tol=NULL, use_seed=FALSE, seed_count=0, n_cluster=1, verbose=FALSE, progress_bar=FALSE)
```
Arguments

- **model**: a R function implementing the model to be simulated. It must take as arguments a vector of model parameter values and it must return a vector of summary statistics. When using the option `use_seed`=`TRUE`, `model` must take as arguments a vector containing a seed value and the model parameter values. A tutorial is provided in the package’s vignette to dynamically link a binary code to a R function. Users may alternatively wish to wrap their binary executables using the provided functions `binary_model` and `binary_model_cluster`. The use of these functions is associated with slightly different constraints on the design of the binary code (see `binary_model` and `binary_model_cluster`).

- **prior**: a list of prior information. Each element of the list corresponds to a model parameter. The list element must be a vector whose first argument determines the type of prior distribution: possible values are "unif" for a uniform distribution on a segment, "normal" for a normal distribution, "lognormal" for a lognormal distribution or "exponential" for an exponential distribution. The following arguments of the list elements contain the characteristics of the prior distribution chosen: for "unif", two numbers must be given: the minimum and maximum values of the uniform distribution; for "normal", two numbers must be given: the mean and standard deviation of the normal distribution; for "lognormal", two numbers must be given: the mean and standard deviation on the log scale of the lognormal distribution; for "exponential", one number must be given: the rate of the exponential distribution. User-defined prior distributions can also be provided. See the vignette for additional information on this topic.

- **nb_design_pts**: a positive integer equal to the desired number of simulations of the model used to build the emulator.

- **nb_simul**: a positive integer equal to the desired number of simulations of the emulator.

- **prior_test**: a string expressing the constraints between model parameters. This expression will be evaluated as a logical expression, you can use all the logical operators including "<", ">", ... Each parameter should be designated with "x1", "x2", ... in the same order as in the prior definition. If not provided, no constraint will be applied.

- **summary_stat_target**: a vector containing the targeted (observed) summary statistics. If not provided, `abc_rejection` only launches the simulations and outputs the simulation results.

- **emulator_span**: a positive number, the number of design points selected for the local regression. 50 by default.

- **tol**: tolerance, a strictly positive number (between 0 and 1) indicating the proportion of simulations retained nearest the targeted summary statistics.

- **use_seed**: logical. If `FALSE` (default), `abc_rejection` provides as input to the function `model` a vector containing the model parameters used for the simulation. If `TRUE`, `abc_rejection` provides as input to the function `model` a vector containing an integer seed value and the model parameters used for the simulation. In this last case, the seed value should be used by `model` to initialize its pseudo-random number generators (if `model` is stochastic).
seed_count  a positive integer, the initial seed value provided to the function model (if use_seed=TRUE). This value is incremented by 1 at each call of the function model.

n_cluster  a positive integer. If larger than 1 (the default value), ABC_rejection will launch model simulations in parallel on n_cluster cores of the computer.

verbose  logical. FALSE by default. If TRUE, ABC_rejection writes in the current directory intermediary results at the end of each step of the algorithm in the file "output". These outputs have a matrix format, in which each row is a different simulation, the first columns are the parameters used for this simulation, and the last columns are the summary statistics of this simulation.

progress_bar  logical, FALSE by default. If TRUE, ABC_rejection will output a bar of progress with the estimated remaining computing time. Option not available with multiple cores.

Value

The returned value is a list containing the following components:

param  The model parameters used in the model simulations.

stats  The summary statistics obtained at the end of the model simulations.

weights  The weights of the different model simulations. In the standard rejection scheme, all model simulations have the same weights.

stats_normalization  The standard deviation of the summary statistics across the model simulations.

nsim  The number of model simulations performed.

nrec  The number of retained simulations (if targeted summary statistics are provided).

computime  The computing time to perform the simulations.

Author(s)

Franck Jabot, Thierry Faure and Nicolas Dumoulin

References


See Also

binary_model, binary_model_cluster, ABC_sequential, ABC_mcmc

Examples

```r
## Not run:

##### EXAMPLE 1 #####

```

```
ABC_mcmc

Coupled to MCMC schemes for ABC

Description

This function implements three different algorithms to perform coupled to MCMC ABC.

Usage

ABC_mcmc(method, model, prior, summary_stat_target, prior_test=NULL, n_rec=100, n_between_sampling=10, n_cluster=1, use_seed = FALSE, verbose = FALSE, dist_weights=NULL, ...)

Arguments

method          a character string indicating the ABC-MCMC algorithm to be used. Possible values are "Marjoram_original", "Marjoram" and "Wegmann". Note that the method "Marjoram_original" cannot be used with multiple cores.

model           a R function implementing the model to be simulated. It must take as arguments a vector of model parameter values and it must return a vector of summary statistics. When using the option use_seed=TRUE, model must take as arguments a vector containing a seed value and the model parameter values. A tutorial is provided in the package's vignette to dynamically link a binary code to a R function. Users may alternatively wish to wrap their binary executables using the provided functions binary_model and binary_model_cluster. The use of these functions is associated with slightly different constraints on the design of the binary code (see binary_model and binary_model_cluster).
prior
a list of prior information. Each element of the list corresponds to a model parameter. The list element must be a vector whose first argument determines the type of prior distribution: possible values are "unif" for a uniform distribution on a segment, "normal" for a normal distribution, "lognormal" for a lognormal distribution or "exponential" for an exponential distribution. The following arguments of the list elements contain the characteristics of the prior distribution chosen: for "unif", two numbers must be given: the minimum and maximum values of the uniform distribution; for "normal", two numbers must be given: the mean and standard deviation of the normal distribution; for "lognormal", two numbers must be given: the mean and standard deviation on the log scale of the lognormal distribution; for "exponential", one number must be given: the rate of the exponential distribution. User-defined prior distributions can also be provided. See the vignette for additional information on this topic.

summary_stat_target
a vector containing the targeted (observed) summary statistics.

prior_test
a string expressing the constraints between model parameters. This expression will be evaluated as a logical expression, you can use all the logical operators including "<", ">". . . Each parameter should be designated with "x1", "x2", . . . in the same order as in the prior definition. If not provided, no constraint will be applied.

n_rec
a positive integer equal to the desired number of sampled points along the MCMC.

n_between_sampling
a positive integer equal to the desired spacing between sampled points along the MCMC.

n_cluster
a positive integer. If larger than 1 (the default value), abc_mcmc will launch model simulations in parallel on n_cluster cores of the computer.

use_seed
logical. If FALSE (default), abc_mcmc provides as input to the function model a vector containing the model parameters used for the simulation. If TRUE, abc_mcmc provides as input to the function model a vector containing an integer seed value and the model parameters used for the simulation. In this last case, the seed value should be used by model to initialize its pseudo-random number generators (if model is stochastic).

verbose
logical. FALSE by default. If TRUE, abc_mcmc writes in the current directory intermediary results at the end of each step of the algorithm in the file "output_mcmc". These outputs have a matrix format, in which each row is a different simulation, the first columns are the parameters used for this simulation, the following columns are the summary statistics of this simulation, and the last column is the distance between the simulation and the data.

dist_weights
a vector containing the weights to apply to the distance between the computed and the targeted statistics. These weights can be used to give more importance to a summary statistic for example. The weights will be normalized before applying them. If not provided, no weights will be applied.

... Additional arguments can be passed depending on the chosen method (see below)
Details

See the package’s vignette for details on ABC-MCMC.

Value

The returned value is a list containing the following components:

- `param`: The model parameters used in the model simulations.
- `stats`: The summary statistics obtained at the end of the model simulations.
- `dist`: The distance of the simulations to the data.
- `stats_normalization`: The standard deviation of the summary statistics across the model simulations of the initial step. These values are used to normalize the summary statistics before the computation of the Euclidean distance between simulations and data. If method is "Marjoram_original", this is equal to `tab_normalization`. If method is "Wegmann", this is not provided.
- `epsilon`: The final maximal distance between simulations and data in the retained sample of particles.
- `nsim`: The number of model simulations performed.
- `n_between_sampling`: The spacing between two sampled points in the MCMC.
- `computime`: The computing time to perform the simulations.
- `min_stats`: The minimal values of each summary statistics during the calibration step, given when method is "Wegmann".
- `max_stats`: The maximal values of each summary statistics during the calibration step, given when method is "Wegmann".
- `lambda`: The lambda values of the Box-Cox transformation, given when method is "Wegmann".
- `geometric_mean`: The geometric means, given when method is "Wegmann".
- `boxcox_mean`: The means of Box-Cox transforms, given when method is "Wegmann".
- `boxcox_sd`: The standard deviations of Box-Cox transforms, given when method is "Wegmann".
- `pls_transform`: The matrix of PLS transformation, given when method is "Wegmann".
- `numcomp`: The number of used components for the PLS transformation, given when method is "Wegmann".

Additional parameters

Depending on the choosen method, you can specify the following arguments:

- `dist_max`: a positive number, used when method is "Marjoram_original". This is the tolerance threshold used during the MCMC. If not provided by the user, it is automatically computed as half the distance between the first simulation and the target summary statistics and a warning is printed.
tab_normalization  a vector of the same length as summary_stat_target, used when method is
"Marjoram_original". Each element contains a positive number by which each summary
statistics must be divided before the computation of the Euclidean distance between simulations and data. If not provided by the user, the simulated summary statistics are divided by the
target summary statistics and a warning is printed.

proposal_range  a vector of the same length as the number of model parameters, used when method
is "Marjoram_original". Each element contains a positive number defining the range of
MCMC jumps for each model parameter. If not provided by the user, a default value is used
for each parameter and a warning is printed. The default value is 1/50 of the prior range
for uniform distributions, 1/20 of the standard deviation of the prior distribution for normal
distributions, 1/20 * exp ( sigma * sigma for lognormal distributions where sigma is the stan-
dard deviation of the prior distribution in the log scale, and 1/20 of the inverse of the rate for
exponential distributions.

n_calibration  a positive integer, used when method is "Marjoram" or "Wegmann". This is the
number of simulations performed during the calibration step. Default value is 10000.

tolerance_quantile  a positive number between 0 and 1 (strictly), used when method is "Marjoram"
or "Wegmann". This is the percentage of simulations retained during the calibration step to
determine the tolerance threshold to be used during the MCMC. Default value is 0.01.

proposal_phi  a positive number, used when method is "Marjoram" or "Wegmann". This is a scaling
factor defining the range of MCMC jumps. Default value is 1.

numcomp  a positive integer, used when method is "Wegmann". This is the number of components
to be used for PLS transformations. Default value is 0 which encodes that this number is equal
to the number of summary statistics.

seed_count  a positive integer, the initial seed value provided to the function model (if use_seed=TRUE).
This value is incremented by 1 at each call of the function model.

progress_bar  logical, FALSE by default. If TRUE, ABC_mcmc will output a bar of progression with
the estimated remaining computing time. Option not available with multiple cores.

Author(s)
Franck Jabot, Thierry Faure and Nicolas Dumoulin

References
likelihoods. *PNAS*, 100, 15324–15328.

Wegmann, D., Leuenberger, C. and Excoffier, L. (2009) Efficient approximate Bayesian computa-

See Also
binary_model, binary_model_cluster, ABC_rejection, ABC_emulation, ABC_sequential

Examples

```r
## Not run:

#### EXAMPLE 1 ####
```
### the model has two parameters and outputs two summary statistics.
### defining a simple toy model:
```
toy_model=function(x){ c( x[1] + x[2] + rnorm(1,0,0.1) , x[1] * x[2] + rnorm(1,0,0.1)) }
```

### define prior information
```
toy_prior=list(c("unif",0,1),c("normal",1,2))
```
# a uniform prior distribution between 0 and 1 for parameter 1, and a normal distribution # of mean 1 and standard deviation of 2 for parameter 2.

### define the targeted summary statistics
```
sum_stat_obs=c(1.5,0.5)
```

### to perform the Marjoram et al. (2003)'s method:
```
ABC_Marjoram_original<-ABC_mcmc(method="Marjoram_original", model=toy_model, prior=toy_prior, summary_stat_target=sum_stat_obs)
```

### artificial example to perform the Marjoram et al. (2003)'s method, with modifications
# drawn from Wegmann et al. (2009) without Box-Cox and PLS transformations.
```
ABC_Marjoram<-ABC_mcmc(method="Marjoram", model=toy_model, prior=toy_prior, summary_stat_target=sum_stat_obs)
```

### artificial example to perform the Wegmann et al. (2009)'s method.
```
ABC_Wegmann<-ABC_mcmc(method="Wegmann", model=toy_model, prior=toy_prior, summary_stat_target=sum_stat_obs)
```

##### EXAMPLE 2 ######

### this time, the model is a C++ function packed into a R function -- this time, # the option 'use_seed' must be turned to TRUE.

### define prior information
```
trait_prior=list(c("unif",3,5),c("unif",-2.3,1.6),c("unif",-25,125),c("unif",-0.7,3.2))
```

### define the targeted summary statistics
```
sum_stat_obs=c(100,2.5,20,30000)
```

### artificial example to perform the Marjoram et al. (2003)'s method.
```
n=10
ABC_Marjoram_original<-ABC_mcmc(method="Marjoram_original", model=trait_model,
Description

This function launches a series of nb_simul model simulations with model parameters drawn in the prior distribution specified in prior_matrix.

Usage

```r
ABC_rejection(model, prior, nb_simul, prior_test=NULL, summary_stat_target=NULL, tol=NULL, use_seed=FALSE, seed_count=0, n_cluster=1, verbose=FALSE, progress_bar=FALSE)
```

Arguments

- `model`: a R function implementing the model to be simulated. It must take as arguments a vector of model parameter values and it must return a vector of summary statistics. When using the option `use_seed=TRUE`, `model` must take as arguments a vector containing a seed value and the model parameter values. A tutorial is provided in the package’s vignette to dynamically link a binary code to a R function. Users may alternatively wish to wrap their binary executables using the provided
functions `binary_model` and `binary_model_cluster`. The use of these functions is associated with slightly different constraints on the design of the binary code (see `binary_model` and `binary_model_cluster`).

**prior**
a list of prior information. Each element of the list corresponds to a model parameter. The list element must be a vector whose first argument determines the type of prior distribution: possible values are "unif" for a uniform distribution on a segment, "normal" for a normal distribution, "lognormal" for a lognormal distribution or "exponential" for an exponential distribution. The following arguments of the list elements contain the characteristics of the prior distribution chosen: for "unif", two numbers must be given: the minimum and maximum values of the uniform distribution; for "normal", two numbers must be given: the mean and standard deviation of the normal distribution; for "lognormal", two numbers must be given: the mean and standard deviation on the log scale of the lognormal distribution; for "exponential", one number must be given: the rate of the exponential distribution. User-defined prior distributions can also be provided. See the vignette for additional information on this topic.

**nb_simul**
a positive integer equal to the desired number of simulations of the model.

**prior_test**
a string expressing the constraints between model parameters. This expression will be evaluated as a logical expression, you can use all the logical operators including "<", ">", etc... Each parameter should be designated with "x1", "x2", etc... in the same order as in the prior definition. If not provided, no constraint will be applied.

**summary_stat_target**
a vector containing the targeted (observed) summary statistics. If not provided, `ABC_rejection` only launches the simulations and outputs the simulation results.

**tol**
tolerance, a strictly positive number (between 0 and 1) indicating the proportion of simulations retained nearest the targeted summary statistics.

**use_seed**
logical. If FALSE (default), `ABC_rejection` provides as input to the function `model` a vector containing the model parameters used for the simulation. If TRUE, `ABC_rejection` provides as input to the function `model` a vector containing an integer seed value and the model parameters used for the simulation. In this last case, the seed value should be used by `model` to initialize its pseudo-random number generators (if `model` is stochastic).

**seed_count**
a positive integer, the initial seed value provided to the function `model` (if use.seed=TRUE). This value is incremented by 1 at each call of the function `model`.

**n_cluster**
a positive integer. If larger than 1 (the default value), `ABC_rejection` will launch `model` simulations in parallel on `n_cluster` cores of the computer.

**verbose**
logical. FALSE by default. If TRUE, `ABC_rejection` writes in the current directory intermediary results at the end of each step of the algorithm in the file "output". These outputs have a matrix format, in which each raw is a different simulation, the first columns are the parameters used for this simulation, and the last columns are the summary statistics of this simulation.

**progress_bar**
logical, FALSE by default. If TRUE, `ABC_rejection` will output a bar of progress with the estimated remaining computing time. Option not available with multiple cores.
Value

The returned value is a list containing the following components:

- `param`: The model parameters used in the model simulations.
- `stats`: The summary statistics obtained at the end of the model simulations.
- `weights`: The weights of the different model simulations. In the standard rejection scheme, all model simulations have the same weights.
- `stats_normalization`: The standard deviation of the summary statistics across the model simulations.
- `nsim`: The number of model simulations performed.
- `nrec`: The number of retained simulations (if targeted summary statistics are provided).
- `computime`: The computing time to perform the simulations.

Author(s)

Franck Jabot, Thierry Faure and Nicolas Dumoulin

References


See Also

- `binary_model`, `binary_model_cluster`, `ABC_sequential`, `ABC_mcmc`

Examples

```r
### EXAMPLE 1 ###
set.seed(1)

## artificial example to show how to use the 'ABC_rejection' function.
## defining a simple toy model:
toy_model<-function(x){ 2 * x + 5 + rnorm(1,0,0.1) }

## define prior information
toy_prior=list(c("unif",0,1)) # a uniform prior distribution between 0 and 1

## only launching simulations with parameters drawn in the prior distributions
set.seed(1)
n=10
ABC_sim<-ABC_rejection(model=toy_model, prior=toy_prior, nb_simul=n)
ABC_sim

## launching simulations with parameters drawn in the prior distributions
# and performing the rejection step
```
sum_stat_obs=6.5
tolerance=0.2
ABC_rej<-ABC_rejection(model=toy_model, prior=toy_prior, nb_simul=n,
  summary_stat_target=sum_stat_obs, tol=tolerance)

## NB: see the package's vignette to see how to pipeline 'ABC_rejection' with the function
# 'abc' of the package 'abc' to perform other rejection schemes.
## Not run:

####### EXAMPLE 2 #######
#########################################################

## this time, the model has two parameters and outputs two summary statistics.
## defining a simple toy model:
toy_model2<-function(x){
c(x[1] + x[2] + rnorm(1,0,0.1), x[1]*x[2] + rnorm(1,0,0.1))
}

## define prior information
toy_prior2=list(c("unif",0,1),c("normal",1,2))
# a uniform prior distribution between 0 and 1 for parameter 1, and a normal distribution
# of mean 1 and standard deviation of 2 for parameter 2.

## only launching simulations with parameters drawn in the prior distributions
set.seed(1)
n=10
ABC_sim<-ABC_rejection(model=toy_model2, prior=toy_prior2, nb_simul=n)
ABC_sim

## launching simulations with parameters drawn in the prior distributions
# and performing the rejection step
sum_stat_obs2=c(1.5,0.5)
tolerance=0.2
ABC_rej<-ABC_rejection(model=toy_model2, prior=toy_prior2, nb_simul=n,
  summary_stat_target=sum_stat_obs2, tol=tolerance)

## NB: see the package's vignette to see how to pipeline 'ABC_rejection' with the function
# 'abc' of the package 'abc' to perform other rejection schemes.

####### EXAMPLE 3 #######
#########################################################

## this time, the model is a C++ function packed into a R function -- this time, the option
# 'use_seed' must be turned to TRUE.
n=10
trait_prior=list(c("unif",3,5),c("unif",-2,3,1.6),c("unif",-25,125),c("unif",-0.7,3.2))

trait_prior

## only launching simulations with parameters drawn in the prior distributions
ABC_sim<-ABC_rejection(model=trait_model, prior=trait_prior, nb_simul=n, use_seed=TRUE)
ABC_sim

## launching simulations with parameters drawn in the prior distributions and performing
# the rejection step
**ABC_sequential**

Sequential sampling schemes for ABC

**Description**

This function implements four different algorithms to perform sequential sampling schemes for ABC. Sequential sampling schemes consist in sampling initially model parameters in the prior distribution, just like in a standard rejection-based ABC, in order to obtain a rough posterior distribution of parameter values, and in subsequently sampling close to this rough posterior distribution to refine it. Sequential sampling schemes have been shown to be more efficient than standard rejection-based procedures.

**Usage**

```r
ABC_sequential(method, model, prior, nb_simul, summary_stat_target,
    prior_test=NULL, n_cluster = 1, use_seed = FALSE, verbose = FALSE,
    dist_weights=NULL, ...)```

```r
sum_stat_obs=c(100,2.5,20,30000)
tolerance=0.2
ABC_rej<-ABC_rejection(model=trait_model, prior=trait_prior, nb_simul=n,
    summary_stat_target=sum_stat_obs, tol=tolerance, use_seed=TRUE)
```

## NB: see the package's vignette to see how to pipeline 'ABC_rejection' with the function # 'abc' of the package 'abc' to perform other rejection schemes.

#### EXAMPLE 4 - Parallel implementations ####

##### NB: the option use_seed must be turned to TRUE.#####

## For models already running with the option use_seed=TRUE, simply change # the value of n_cluster:

```r
sum_stat_obs=c(100,2.5,20,30000)
ABC_simb<-ABC_rejection(model=trait_model, prior=trait_prior, nb_simul=n,
    use_seed=TRUE, n_cluster=2)
```

## For other models, change the value of n_cluster and modify the model so that the first # parameter becomes a seed information value:

```r
toy_model_parallel<function(x){
    set.seed(x[1])
    2 * x[2] + 5 + rnorm(1,0,0.1) }
    sum_stat_obs=6.5
    ABC_simb<-ABC_rejection(model=toy_model_parallel, prior=toy_prior, nb_simul=n,
        use_seed=TRUE, n_cluster=2)
```

## End(Not run)
Arguments

method  
a character string indicating the sequential algorithm to be used. Possible values are "Beaumont", "Drovandi", "Delmoral", "Lenormand" and "Emulation".

model  
a R function implementing the model to be simulated. It must take as arguments a vector of model parameter values and it must return a vector of summary statistics. When using the option use_seed=TRUE, model must take as arguments a vector containing a seed value and the model parameter values. A tutorial is provided in the package’s vignette to dynamically link a binary code to a R function. Users may alternatively wish to wrap their binary executables using the provided functions binary_model and binary_model_cluster. The use of these functions is associated with slightly different constraints on the design of the binary code (see binary_model and binary_model_cluster).

prior  
a list of prior information. Each element of the list corresponds to a model parameter. The list element must be a vector whose first argument determines the type of prior distribution: possible values are "unif" for a uniform distribution on a segment, "normal" for a normal distribution, "lognormal" for a lognormal distribution or "exponential" for an exponential distribution. The following arguments of the list elements contain the characteristics of the prior distribution chosen: for "unif", two numbers must be given: the minimum and maximum values of the uniform distribution; for "normal", two numbers must be given: the mean and standard deviation of the normal distribution; for "lognormal", two numbers must be given: the mean and standard deviation on the log scale of the lognormal distribution; for "exponential", one number must be given: the rate of the exponential distribution. Note that when using the method "Lenormand", solely uniform prior distributions are supported. User-defined prior distributions can also be provided. See the vignette for additional information on this topic.

nb_simul  
a positive integer equal to the desired number of simulations of the model below the tolerance threshold when method is "Beaumont", "Drovandi" and "Delmoral". When method is "Lenormand", the number of simulations below the tolerance threshold is equal to nb_simul * alpha. See the package’s vignette and Lenormand et al. (2012) for details.

summary_stat_target  
a vector containing the targeted (observed) summary statistics.

prior_test  
a string expressing the constraints between model parameters. This expression will be evaluated as a logical expression, you can use all the logical operators including "<", ">", ..., Each parameter should be designated with "X1", "X2", ... in the same order as in the prior definition. If not provided, no constraint will be applied.

n_cluster  
a positive integer. If larger than 1 (the default value), ABC_sequential will launch model simulations in parallel on n_cluster cores of the computer.

use_seed  
logical. If FALSE (default), ABC_sequential provides as input to the function model a vector containing the model parameters used for the simulation. If TRUE, ABC_sequential provides as input to the function model a vector containing an integer seed value and the model parameters used for the simulation. In this last case, the seed value should be used by model to initialize its pseudo-random number generators (if model is stochastic).
verbose logical. FALSE by default. If TRUE, ABC_sequential writes in the current directory intermediary results at the end of each step of the algorithm various files. The file "n_simul_tot_step_iteration" (where iteration is the step number) contains the total number of simulations performed since the beginning of the algorithm at the end of the step "iteration". The file "R_step_iteration" (when using the method "Drovandi") is the parameter R used during the step "iteration" (see Drovandi and Pettitt 2011 for details). The file "p_acc_iteration" (when using the method "Lenormand") is the parameter p_acc computed at the end of the step "iteration" (see Lenormand et al. 2012 for details). The file "tolerance_step_iteration" (when using the method "Drovandi", "Delmoral" or "Lenormand") is the tolerance computed at the end of the step "iteration". The file "output_step_iteration" gives the simulations kept after each iteration and has a matrix format, in which each row is a different simulation, the first column is the weight of the simulation, the following columns are the parameters used for this simulation, and the last columns are the summary statistics of this simulation. The file "model_step_iteration" gives the simulations performed at each iteration and has a matrix format, in which each row is a different simulation, the first column is the weight of the simulation, the following columns are the parameters used for this simulation, and the last columns are the summary statistics of this simulation. All these informations are further stored in a list (with the same formats) and are accessible from R - see intermediary in the value section below.

dist_weights a vector containing the weights to apply to the distance between the computed and the targeted statistics. These weights can be used to give more importance to a summary statistic for example. The weights will be normalized before applying them. If not provided, no weights will be applied.

Additional arguments can be passed depending on the choosen method (see below)

Details

See the package’s vignette for details on the four algorithms.

Value

The returned value is a list containing the following components:

param The model parameters used in the model simulations.
stats The summary statistics obtained at the end of the model simulations.
weights The weights of the different model simulations.
stats_normalization The standard deviation of the summary statistics across the model simulations of the initial step. These values are used to normalize the summary statistics before the computation of the Euclidean distance between simulations and data.
epsilon The final maximal distance between simulations and data in the retained sample of particles.
nsim The number of model simulations performed.
The computing time to perform the simulations.

Intermediary results stored when the option "verbose=TRUE" is chosen. Each element of this list corresponds to a different step. See the argument `verbose` above for more details on the information stored.

Additional parameters

Depending on the choosen method, you can specify the following arguments:

- **seed_count** a positive integer, the initial seed value provided to the function `model` (if `use_seed=TRUE`). This value is incremented by 1 at each call of the function `model`.

- **inside_prior** logical used when method is "Beaumont", "Lenormand" or "Emulation". TRUE by default. If FALSE, parameter sampling is not restricted to the initial ranges of the prior distribution during the subsequent algorithm steps.

- **tolerance_tab** a vector containing the sequence of tolerance thresholds when method is "Beaumont", or the targeted final tolerance threshold when method is "Drovandi".

- **alpha** a positive number between 0 and 1 (strictly) used when method is "Drovandi", "Delmoral", "Lenormand" or "Emulation". alpha is the proportion of particles rejected at each step in the algorithm "Drovandi". This is the proportion of particles kept at each step in the algorithms "Delmoral", "Lenormand" and "Emulation". Default values are 0.5 when method is "Drovandi", "Lenormand" or "Emulation" and 0.9 for "Delmoral". See the package’s vignette for details.

- **c** a positive number between 0 and 1 (strictly) used when method is "Drovandi". This is the expected proportion of particles which are going to be duplicated at each step. Default value is 0.01. See the package’s vignette and Drovandi and Pettitt (2011) for details.

- **first_tolerance_level_auto** logical used when method is "Drovandi". Default value is TRUE. In this case, the first tolerance threshold is determined by the algorithm, by taking the 1-alpha quantile of the distances between the simulated and targeted summary statistics. If FALSE, the initial tolerance threshold for the first step has to be provided as the first element of the vector `tolerance_tab`. In this case, the targeted final tolerance threshold is the second element of `tolerance_tab`.

- **M** a positive integer used when method is "Delmoral". This is the number of model simulations performed for each parameter set. Default value is 1. See the package’s vignette and Del Moral et al. (2012) for details.

- **nb_threshold** a positive integer used when method is "Delmoral". Default value is 0.5*nb_simul. This is the minimal effective sample size below which a resampling step is launched. See the package’s vignette and Del Moral et al. (2012) for details.

- **tolerance_target** a positive number used when method is "Delmoral". This is the targeted final tolerance threshold.

- **p_acc_min** a positive number between 0 and 1 (strictly) used when method is "Lenormand" or "Emulation". This is the stopping criterion of the algorithm: a small number ensures a better convergence of the algorithm, but at a cost in computing time. Default value is 0.05. See the package’s vignette and Lenormand et al. (2012) for details.

- **n_step_emulation** a positive integer, the number of times the emulation is repeated. 9 by default.

- **emulator_span** a positive number, the number of design points selected for the local regression. 50 by default.
progress_bar logical. FALSE by default. If TRUE, ABC_sequential will output a bar of progression with the estimated remaining computing time. Option not available with multiple cores.

Author(s)
Franck Jabot, Thierry Faure and Nicolas Dumoulin

References

See Also
*binary_model, binary_model_cluster, ABC_rejection, ABC_emulation, ABC_mcmc*

Examples
```r
## Not run:
#### EXAMPLE 1 ####
set.seed(1)

## artificial example to show how to use the 'ABC_sequential' function.
## defining a simple toy model:
toy_model<-function(x){ 2 * x + 5 + rnorm(1,0,0.1) }

## define prior information
toy_prior=list(c("unif",0,1)) # a uniform prior distribution between 0 and 1

## define the targeted summary statistics
sum_stat_obs=6.5

## to perform the Beaumont et al. (2009)'s method:
##
tolerance=c(1.5,0.5)
ABC_Beaumont<-ABC_sequential(method="Beaumont", model=toy_model, prior=toy_prior, nb_simul=20, summary_stat_target=sum_stat_obs, tolerance_tab=tolerance)

## to perform the Drovandi and Pettitt (2011)'s method:
##
```
tolerance=0.5
c_drov=0.7
ABC_Drovandi<-ABCSequential(method="Drovandi", model=toy_model, prior=toy_prior,
  nb_simul=20, summary_stat_target=sum_stat_obs, tolerance_tab=tolerance, c=c_drov)
ABC_Drovandi

### to perform the Del Moral et al. (2012)'s method:
###
alpha_delmo=0.5
tolerance=0.5
ABC_Delmoral<-ABCSequential(method="Delmoral", model=toy_model, prior=toy_prior,
  nb_simul=20, summary_stat_target=sum_stat_obs, alpha=alpha_delmo, tolerance_target=tolerance)
ABC_Delmoral

### to perform the Lenormand et al. (2012)'s method:
###
p_acc=0.4
ABC_Lenormand<-ABCSequential(method="Lenormand", model=toy_model, prior=toy_prior,
  nb_simul=20, summary_stat_target=sum_stat_obs, p_acc_min=p_acc)
ABC_Lenormand

##### EXAMPLE 2 ######

```r
# this time, the model has two parameters and outputs two summary statistics.
# defining a simple toy model:
toy_model2<-function(x) { c(x[1] + x[2] + rnorm(1,0,0.1), x[1] * x[2] + rnorm(1,0,0.1)) }

# define prior information
toy_prior2=list(c('unif',0,1),c("normal",1,2))
# a uniform prior distribution between 0 and 1 for parameter 1, and a normal distribution
# of mean 1 and standard deviation of 2 for parameter 2.

# define the targeted summary statistics
sum_stat_obs2=c(1.5,0.5)

# to perform the Beaumont et al. (2009)'s method:
#
tolerance=c(1.5,0.5)
ABC_Beaumont<-ABCSequential(method="Beaumont", model=toy_model2, prior=toy_prior2,
  nb_simul=20, summary_stat_target=sum_stat_obs2, tolerance_tab=tolerance)
ABC_Beaumont

# to perform the Drovandi and Pettitt (2011)'s method:
#
tolerance=0.5
c_drov=0.7
ABC_Drovandi2<-ABCSequential(method="Drovandi", model=toy_model2, prior=toy_prior2,
  nb_simul=20, summary_stat_target=sum_stat_obs2, tolerance_tab=tolerance, c=c_drov)
ABC_Drovandi2

# to perform the Del Moral et al. (2012)'s method:
## Example

### Example 3

```r
## This time, the model is a C++ function packed into a R function -- this time, the option
## 'use_seed' must be turned to TRUE.

n=10
# define prior information
trait_prior=list(c("unif",3,5),c("unif",-2.3,1.6),c("unif",-25,125),c("unif",0.7,3.2))
# define the targeted summary statistics
sum_stat_obs=c(100,2.5,20,30000)

## to perform the Beaumont et al. (2009)'s method:
#
tolerance=c(8,5)
ABC_Beaumont<-ABC_sequential(method="Beaumont", model=trait_model, prior=trait_prior, nb_simul=20, summary_stat_target=sum_stat_obs, tolerance_tab=tolerance, use_seed=TRUE)
ABC_Beaumont

## to perform the Drovandi and Pettitt (2011)'s method:
#
tolerance=3
c_drov=0.7
ABC_Drovandi<-ABC_sequential(method="Drovandi", model=trait_model, prior=trait_prior, nb_simul=20, summary_stat_target=sum_stat_obs, tolerance_tab=tolerance, c=c_drov, use_seed=TRUE)
ABC_Drovandi

## to perform the Del Moral et al. (2012)'s method:
#
alpha_delmo=0.5
tolerance=3
ABC_Delmoral<-ABC_sequential(method="Delmoral", model=trait_model, prior=trait_prior, nb_simul=20, summary_stat_target=sum_stat_obs, alpha=alpha_delmo, tolerance_target=tolerance, use_seed=TRUE)
ABC_Delmoral

## to perform the Lenormand et al. (2012)'s method:
#
pacc=0.4
# Only uniform priors are supported for the method "Lenormand" (since it performs a Latin
# Hypercube sampling at the beginning):
# a uniform prior distribution between 0 and 1 for parameter 1, and a normal distribution of
# mean 1 and standard deviation of 1 for parameter 2.
ABC_Lenormand<-ABC_sequential(method="Lenormand", model=toy_model2, prior=toy_prior2, nb_simul=20, summary_stat_target=sum_stat_obs2, p_acc_min=pacc)
ABC_Lenormand

```
```r
nb_simul=20, summary_stat_target=sum_stat_obs, alpha=alpha_delmo,
tolerance_target=tolerance, use_seed=TRUE)
ABC_Delmoral

## to perform the Lenormand et al. (2012)'s method:
##
pacc=0.4
ABC_Lenormand<-ABC_sequential(method="Lenormand", model=trait_model, prior=trait_prior,
   nb_simul=20, summary_stat_target=sum_stat_obs, p_acc_min=pacc, use_seed=TRUE)
ABC_Lenormand

#### EXAMPLE 4 - Parallel implementations ####

# NB: the option use_seed must be turned to TRUE.

# For models already running with the option use_seed=TRUE, simply change
# the value of n_cluster:
sum_stat_obs=c(100,2.5,20,30000)
ABC_Lenormand<-ABC_sequential(method="Lenormand", model=trait_model, prior=trait_prior,
   nb_simul=20, summary_stat_target=sum_stat_obs, p_acc_min=pacc, use_seed=TRUE, n_cluster=2)
ABC_Lenormand

# For other models, change the value of n_cluster and modify the model so that the
# first parameter becomes a seed information value:
toy_model_parallel<-function(x){
  set.seed(x[1])
  2 * x[2] + 5 + rnorm(1,0,0.1) }
  sum_stat_obs=6.5
ABC_Lenormand<-ABC_sequential(method="Lenormand", model=toy_model_parallel, prior=toy_prior,
   nb_simul=20, summary_stat_target=sum_stat_obs, p_acc_min=pacc, use_seed=TRUE, n_cluster=2)
ABC_Lenormand

## End(Not run)
```

---

**binary_model**

*Wrapper for a binary executable for non-parallel simulations*

---

**Description**

This function enables to link a binary executable to a `R` function.

**Usage**

`binary_model(command)`
**Arguments**

- **command**: a character string indicating the command to launch the executable on your system.

**Details**

A binary executable used with `binary_model` has to be placed in the current directory of the R session. It further has to respect several constraints: it has to read the seed for its pseudo-random number generator and the model parameters in a file "input", and it must write the summary statistic in a file "output". The file "input" will be generated by the wrapper `binary_model` in the current directory of the R session, and the wrapper will read the file "output" generated by the binary executable. In the file "input", the first line contains the seed, and each subsequent line contains one model parameter value. In the file "output", each summary statistic should be separated by a space or a tab separation. This wrapper should be used for use with a single core of the computer. If the user wishes to use several cores of the computer, the wrapper `binary_model_cluster` should be used. Note that the files "input" and "output" are deleted by the wrapper at the end of the function.

**Value**

A R function wrapping the binary executable, to be used with the EasyABC functions.

**Author(s)**

Franck Jabot, Thierry Faure and Nicolas Dumoulin

**See Also**

`binary_model_cluster, ABC_rejection, ABC_sequential, ABC_mcmc`

**Examples**

```r
## Not run:
## artificial example to show how to use the binary_model function with
## an executable "My_Executable"
ABC_rej<ABC_rejection(model=binary_model("./My_Executable"), prior=..., n_cluster=1,...)

# NB: on windows, "My_Executable" should be of the form "My_Executable.exe" :
ABC_rej<ABC_rejection(model=binary_model("./My_Executable.exe"), prior=..., n_cluster=1,...)

## End(Not run)
```
### Description

This function enables to link a binary executable to a R function.

### Usage

```r
binary_model_cluster(command)
```

### Arguments

- **command**: a character string indicating the command to launch the executable on your system.

### Details

A binary executable used with `binary_model_cluster` has to be placed in the current directory of the R session. It further has to respect several constraints: 1- it has to have a single argument: a number \( k \) used by the binary executable to know in which files read and write. 2- it has to read the seed for its pseudo-random number generator and the model parameters in a file "inputk" (where \( k \) is the argument passed to the executable: input1, input2,...). 3- it has to write the summary statistic in a file "outputk" (where \( k \) is the argument passed to the executable: output1, output2,...). The file "inputk" will be generated by the wrapper `binary_model_cluster` in the current directory of the R session, and the wrapper will read the file "outputk" generated by the binary executable. This construction ensures that each core reads and writes in different files. In the file "inputk", the first line contains the seed, and each subsequent line contains one model parameter value. In the file "outputk", each summary statistic should be separated by a space or a tab separation. This wrapper should be used for use with multiple cores of the computer. If the user wishes to use a single core of the computer, the wrapper `binary_model` should be used. Note that the files "inputk" and "outputk" are deleted by the wrapper at the end of the function.

### Value

A R function wrapping the binary executable, to be used with the EasyABC functions.

### Author(s)

Franck Jabot, Thierry Faure and Nicolas Dumoulin

### See Also

- `binary_model`, `ABC_rejection`, `ABC_sequential`, `ABC_mcmc`
trait_model

Trait model for EasyABC tutorial

Description

A stochastic individual-based model to demonstrate how the EasyABC functions can be used.

Details

This model is drawn from Jabot (2010) A stochastic dispersal-limited trait-based model of community dynamics. Journal of Theoretical Biology, 262, 650-661. It represents the stochastic dynamics of an ecological community where each species is represented by a set of traits (i.e. characteristics) which determine its competitive ability. The model requires four parameters: an immigration rate $i$, and three additional parameters ($h$, $a$ and $\sigma$) describing the way traits determine species competitive ability. The model additionally requires two fixed variables: the total number of individuals in the local community $j$ and the number of traits used $n_t$. The model outputs four summary statistics: the species richness of the community $S$, its Shannon’s index $H$, the mean of the trait value among individuals $MTV$ and the skewness of the trait value distribution $STV$. The simulation code `trait_model` reads sequentially $j$, $i$, $a$, $n_t$, $h$ and $\sigma$.

Examples

```r
trait_model(input=c(1,500,1,1,1,1))
```
Index

*Topic abc
  ABC_emulation, 2
  ABC_mcmc, 5
  ABC_rejection, 10
  ABC_sequential, 14
  binary_model, 21
  binary_model_cluster, 23
  EasyABC-package, 2

*Topic emulation
  ABC_emulation, 2

*Topic inference
  ABC_emulation, 2
  ABC_mcmc, 5
  ABC_rejection, 10
  ABC_sequential, 14
  binary_model, 21
  binary_model_cluster, 23

*Topic mcmc
  ABC_mcmc, 5

*Topic model
  ABC_emulation, 2
  ABC_mcmc, 5
  ABC_rejection, 10
  ABC_sequential, 14
  binary_model, 21
  binary_model_cluster, 23

*Topic package
  EasyABC-package, 2

*Topic population_monte_carlo
  ABC_sequential, 14

*Topic sample
  trait_model, 24

*Topic sequential_monte_carlo
  ABC_sequential, 14

binary_model, 2–5, 8, 11, 12, 15, 18, 21, 23
binary_model_cluster, 2–5, 8, 11, 12, 15, 18, 22, 23

EasyABC (EasyABC-package), 2
EasyABC-package, 2

trait_model, 24